フレーム・メモリを備えた画像処理回路の設計

江崎雅康

「FPGA XC3S500E-VQ208（208ピン）＋2Mバイト高速SRAM基板」と「画像ベースボード」で構成する本格的な画像処理回路の詳細を紹介する.　（編集部）

1　フレーム・メモリ付き画像制御回路で行った実験

第1章で紹介した「XC3S500E-VQ208（208ピン）＋2Mバイト高速SRAM基板」CQ-SP3E208を使って，

- SCCB（Serial Camera Control Bus）インターフェースを介したCMOSカメラの設定
- CMOSカメラからのデータをフレーム・メモリに記録
- フレーム・メモリのデータをアナログVGAに出力
- マイクロプロセッサによる画像ピクセルへのアクセス
- マイクロプロセッサによる画像キャプチャの制御

などを行いました.

本章ではこの画像制御回路の構成を詳しく紹介し，マイクロプロセッサのプログラムもしくはVHDLソースを書き換えることにより各種画像応用システムを構成する助けとします.

第1章で行った実験は，次の3種類です.

①カメラから30フレーム/sで出力した画像データをフレーム・メモリに記録し，同じフレーム・メモリから60フレーム/sで読み出した画像データをアナログVGAに出力し表示する.

②カメラ画像をフレーム・メモリ0，フレーム・メモリ1に記録する．マイクロプロセッサから各フレームのピクセル・データを読み出し，差分データをフレーム・メモリ2に書き込んでアナログVGAに表示するソフトウェア差分処理の実験.

③カメラ画像をフレーム・メモリ0に記録し，次にフレーム・メモリ0から読み出したデータとカメラ・データの差分をFPGAで演算する．その結果をフレーム・メモリ1に記録し，アナログVGAに表示するハードウェア差分処理の実験.

2　画像ベースボード，フレーム・メモリ基板，Interface誌付属SH7144基板で構成した画像処理システム

図1は本誌2007年8月号で紹介した画像ベースボードCQ-SP3EDWの回路図です．この回路図の

- 40ピン・プラグ　J_1
- 34ピン・プラグ　J_2
- 20ピン・プラグ　J_{12}

に第1章で紹介した「XC3S500E-VQ208＋2Mバイト高速SRAM基板」CQ-SP3E208のヘッダJ_3，J_2，J_4を接続します.

制御にはInterface誌2006年6月号付属基板CQ-SH7144を使いました．この基板をCQ-SP3E208の40ピン・プラグJ_1，J_5に接続すると**写真1**に示すように3段重ねになります.

CQ-SP3E208は基板上に48MHzクロック・モジュールを実装していますが，今回の実験ではCQ-SH7144のクロッ

KeyWord　フレーム・メモリ付き画像制御回路，SCCB，デジタルCMOSカメラ，RGB：565モード，CPUによる差分演算処理，ハードウェアによる差分

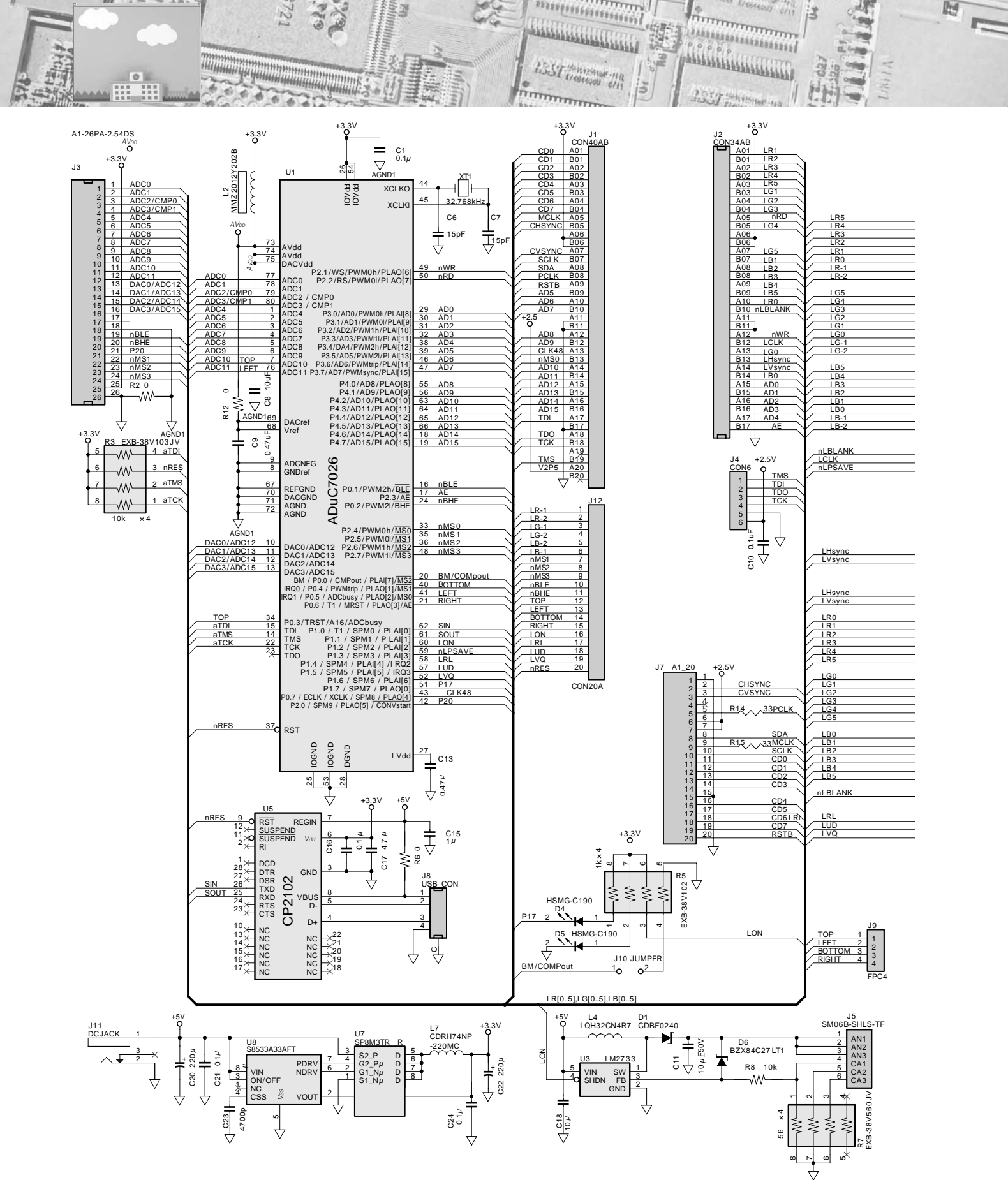
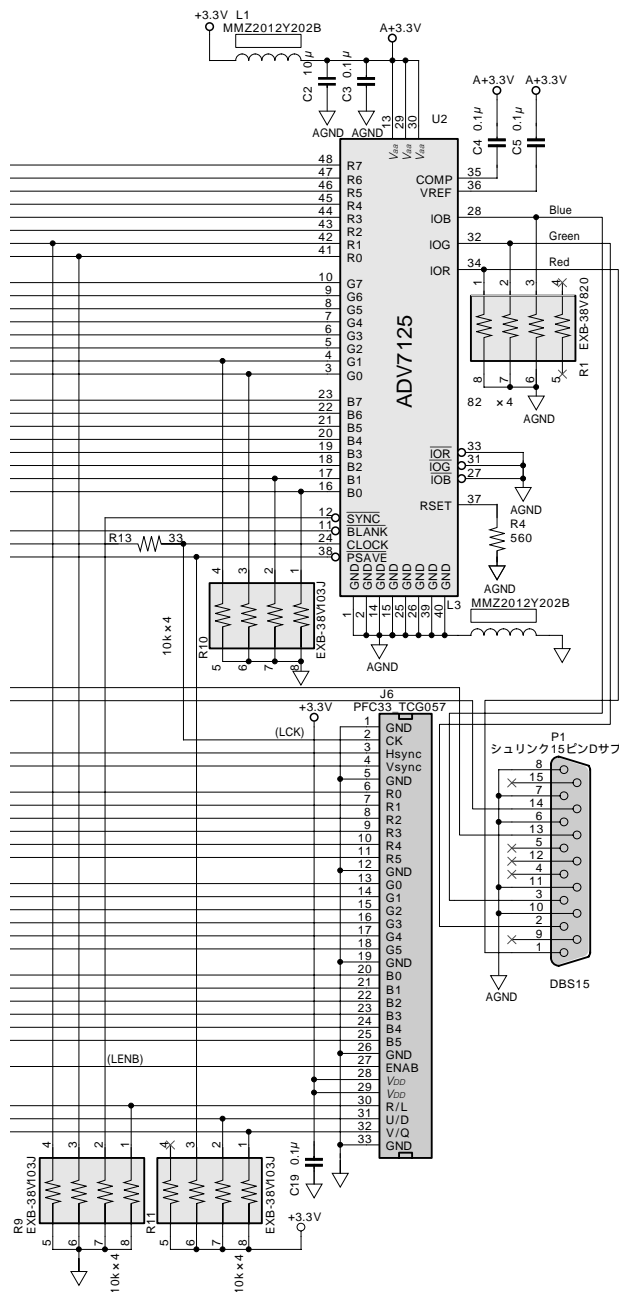



図1　画像ベースボードCQ-SP3EDWの回路図

画像用D-AコンバータADV7125，マイクロプロセッサADuC7026，USB-シリアル変換IC CP2102，液晶バックライト用LED駆動回路，3.3Vスイッチング・レギュレータなどを備える．本誌2007年8月号で紹介したもの．

ク BCLK（48MHz）を使いました．

CPU と FPGA のクロックの位相が一致しているとタイミング設計が容易になります．

Interface 誌付属基板としては 2007 年 5 月号の CQ-V850 もあります．こちらも機会をみて試してみたいと思います．

画像ベースボード上に ADuC7026 が実装されています．SCCB の制御などはこの CPU で行うことも可能ですが，外部メモリ空間は合計 512K バイトしか利用できません．2M バイトのフレーム・メモリを直接アクセスするためには GPIO によるバンク操作などの工夫が必要になります．今回は ADuC7026 は SCCB の制御には使いません．

3 フレーム・メモリを含む画像制御回路の構成

図2 は今回の実験に使ったフレーム・メモリ付き画像回路の構成図です．灰色の部分は FPGA 内部の回路です．

今月号の付属 CD-ROM には VHDL のソース・コードを収録しています．**図3** は実験のため FPGA に実装した回路モジュールの階層構成です．ソース解読の参考になると思います．

回路モジュール FM_REG_CTRL は CPU インターフェースと FPGA の制御レジスタを記述しています．CPU は **図4** に示す制御レジスタを介して画像制御回路をコントロールします．

今回の実験には Interface 誌付属基板 CQ-SH7144 を使い，画像制御レジスタおよびフレーム・メモリは SH7144 の CS2 空間に配置しました．

画像制御レジスタは **図5** のメモリ・マップに示すように

900000 番地～90000B 番地

に配置しました．

2M バイトの画像フレーム・メモリはそれぞれ，

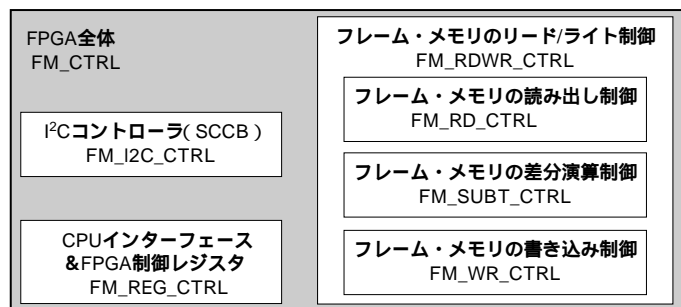



図3　FPGA に実装した回路モジュールの階層構造

4

A00000番地～A95FFF番地　　　　　フレーム0
A96000番地～B2BFFF番地　　　　　フレーム1
B2C000番地～BC1FFF番地　　　　　フレーム2

に配置しています．

図6は画像制御回路の主要なタイミング信号と構成モジュールをブロック図で表示したものです．**図2**の構成図はVHDLソースの記述モジュール間の信号の流れを図示しただけですが，**図6**は制御信号とデータを実際の流れに即して表示しています．

4　デジタルCMOSカメラのインターフェース設定

VGAカメラ・モジュールKBCR-M03VGはSCCBシリ

写真1
画像ベースボード，FPGA＋フレーム・メモリ基板，Interface誌付属SH7144基板で構成した画像処理システム

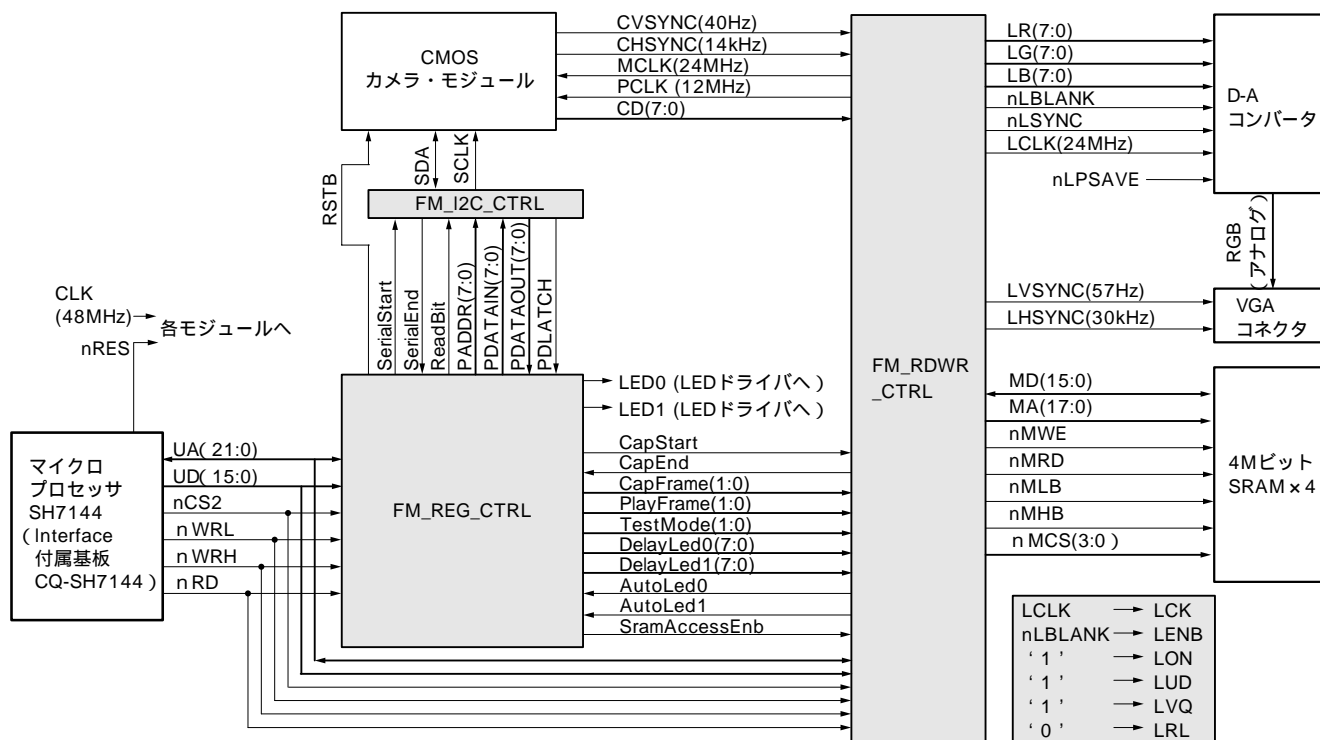
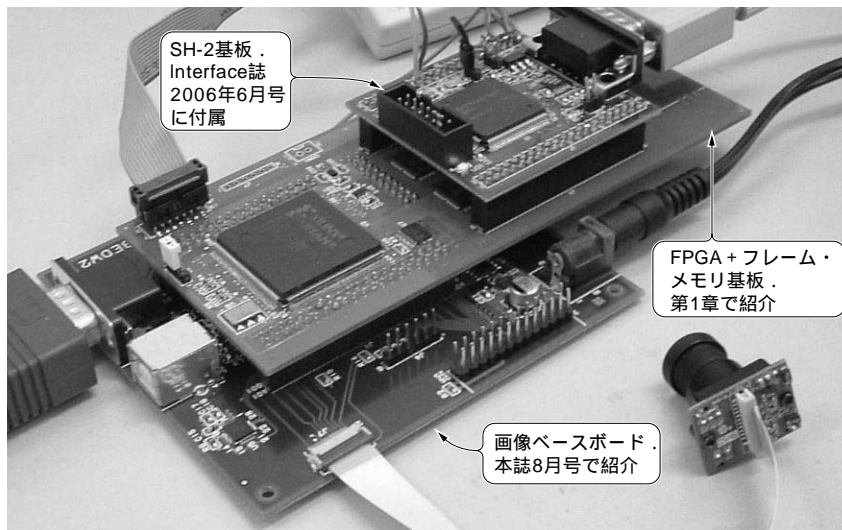
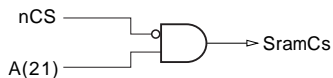
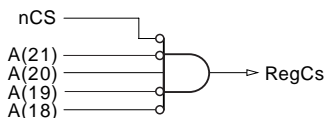
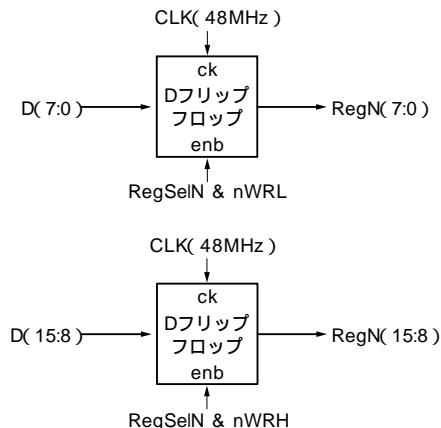
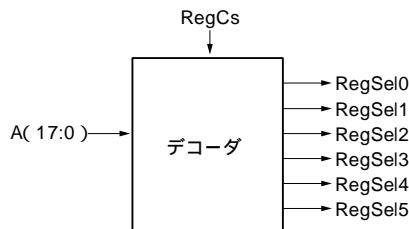



図2　フレーム・メモリを含む画像制御回路の構成図
灰色の部分はFPGAの内部回路

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
デフォルト	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
						Subtract Frame		Subtract Enable	Reserved			Play Frame		Capture Frame		Capture Start

Capture Start ：1を書くと録画スタート．1フレーム・キャプチャ後，自動的に0になる．
Capture Frame ：録画先フレーム・メモリ選択（0，1，2）
Play Frame ：再生元フレーム・メモリ選択（0，1，2）
Subtract Enable：1でフレーム差分をとる．原画像からSubtract Frameの画像を減算する．
Subtract Frame ：原画像から減算するフレーム・メモリを選択する（0，1，2）．

（**a**）0x900000：Capture/Play Control

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
デフォルト	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reserved														Read Bit	Start Bit

Start Bit ：1を書くとシリアル通信（I²C）スタート．通信終了後，自動的に0になる．
Read Bit ：0でシリアル書き込み．1でシリアル読み出し．

（**b**）0x900002：Serial Control

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
デフォルト	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Serial Data								Serial Address							

Serial Address：シリアル通信(I²C)アドレス
Serial Data ：シリアル通信(I²C）データ

（**c**）0x900004：Serial Address/Data

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
デフォルト	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
											SramAccessEnb	RSTB	Test Mode			Reset Start

Reset Start ：テスト用．さしあたり使用しない．
Test Mode ：1で赤一色，2で緑一色，3で青一色をVGAに出力．
RSTB ：1でRSTB端子(CCDカメラのCS端子)がHになる．
SramAccessEnb：1でマイコンからSRAMへの書き込み，読み出しが可能になる．このときCCDカメラからSRAMへの書き込みはできない．

（**d**）0x900006：Miscellaneous

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
デフォルト	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reserved				Input GPIOs				Manual LED Mode	Reserved	Manual LED1	Manual LED0	Output GPIOs			

Output GPIOs ：GPIO0～3に出力する値
Manual LED Mode ：1でManual LED Mode，0でAuto LED Mode(自動発光)
Manual LED0, Manual LED1：1を書くとLED端子が“H”になる(Manual LED Modeが1のとき)
Input GPIOs ：GPIO4～7の状態が入る(GPIO5と6は無効)

（**e**）0x900008：GPIO Control

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
デフォルト	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	LED1 Delay								LED0 Delay							

LED0 Delay：VSYNCからLED0が光るまでのライン数(Auto LED Mode時)
LED1 Delay：VSYNCからLED1が光るまでのライン数(Auto LED Mode時)

（**f**）0x90000A：LED Delay

図4 制御レジスタ(FM_REG_CTRL)
CPUはこの制御レジスタを介して画像制御回路をコントロールする．

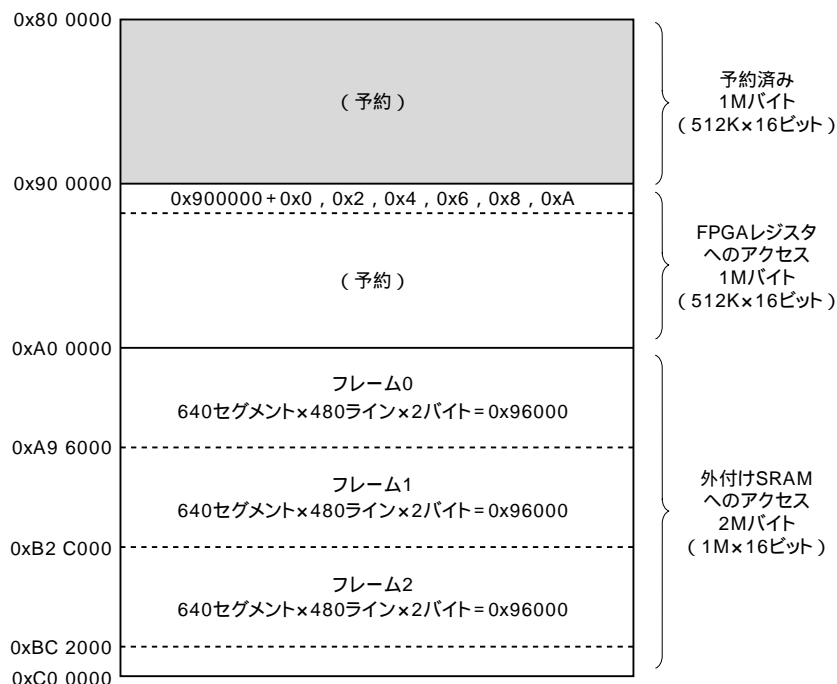



図5
nCS（SHマイコンのCS2出力）でアクセスするときのメモリ・マップ

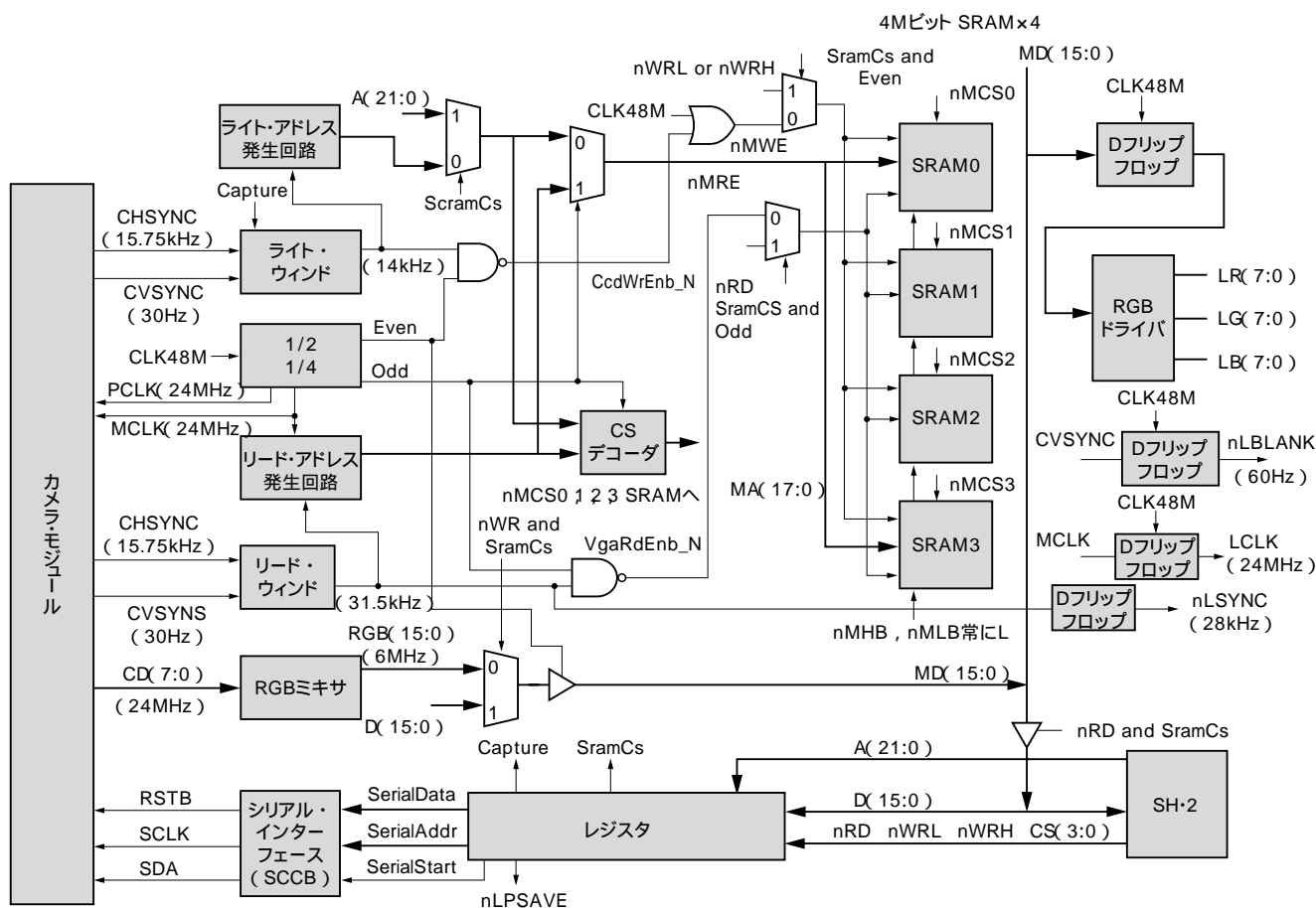



図6　フレーム・メモリを含む画像制御回路の制御ブロック図

FPGAフレーム・メモリ・コントローラ（マイコン⟷SRAMアクセスあり）

アル・インターフェースを介して内部レジスタの設定を行う必要があります．SCCBはI²Cに類似したシリアル通信仕様です．

SCCBは**図2**に示す2本の信号線

- SDA　　データ線（双方向）
- SCLK　　クロック

を介してデータの送受信を行います．

図2の回路 FM_CTL はI²Cの制御回路をハードウェアで備えています．CPUからは**図4(c)**に示す

- シリアル・アドレス/データ・レジスタ　　900004

に必要なアドレスとデータを書き込んだ上で**図4(b)**の

- シリアル制御レジスタ　　900002

のビット0＝Start Bit（Serial Start）を‘1’に設定するだけです．

シリアル通信が完了すると，このビットが‘0’になるのでCPUはこのビット変化をポーリングで待ちます．

リスト1はカメラ画像をアナログRGBディスプレイに動画表示するプログラムです．カメラ・モジュールをRGB：565モードに設定するために

```
SADDT=0x111F；          //RGB：565出力
SCTRL=0x0001；
while((SCTRL&0x0001)==1) {}
```

と記述しています．

カメラ内部レジスタの値を読み出したい時は**図4(b)**を

Read Bit＝1
Start Bit＝1

とします．SCCBシーケンスが終了すると**図4(c)**のレジスタにシリアル・データが格納されるので，CPUから読み出します．

5 デジタルCMOSカメラの画像入力のメモリ制御

3枚の画像フレーム・メモリは録画/表示に関してまったく同じ役割を果たしています．**図4(a)**のCapture/Play Controlレジスタは，画像フレーム・メモリの読み出し/書き込みを制御するレジスタです．

デジタルCMOSカメラの画像をキャプチャ（録画）したい時は，

① 録画先フレーム・メモリをCapture Frame（ビット2～1）で選択した上で
② 録画開始ビット（ビット0）を1に設定

します．

図7はフレーム・メモリ制御のタイミング図です．カメラ・データは24MHzのPCLKに同期して書き込まれます．

RGB：565フォーマットの場合，

- 偶数（even）タイミング　R（5ビット）＋G（3ビット）
- 奇数（odd）タイミング　G（3ビット）＋B（5ビット）

で入力するので，パイプライン処理によりRGB16ビットに変換します．

6 画像フレーム・メモリのタイミング制御

Capture/Play Controlレジスタは，画像の表示も制御します．ビット4とビット3で指定されたフレーム・メモリから，16ビット画像データを，画像用D-AコンバータADV7125とTFT液晶表示回路に送り出します．

画像フレーム・メモリとFPGAは1パスで接続されています．カメラ画像の入力と表示のための読み出しは競合します．この二つのメモリ・アクセスを“見かけ上同時に”行うために，時分割アクセスを行っています．

16ビット画像データをVGA表示するために必要なピクセル・クロック・レートは，24MHzです．メモリのアクセスを48MHzで行うと，2回アクセス可能です．

図7のタイミング図で「SRAMのデータ」にこのしくみを表示しています．

- MCLK＝“L”のタイミングで…カメラ入力
- MCLK＝“H”のタイミングで…VGA画像表示出力

を行っています．

7 画像データのリアルタイム・ハードウェア処理

カメラ・モジュールKBCR-M03VGのVGA出力は30フレーム/sですから，時分割アクセスのタイミングに余裕が出ます．**図7**のタイミング・チャートは，このタイミングを利用して，直前のカメラ・データの読み出し（read back）を行っています．

この機能を利用することにより，フレーム・データのリアルタイム・ハードウェア処理を行うことができます．**図4(a)** Capture/Play Controlレジスタの（Subtract Enable）は，この機能を有効にするビットです．演算対象のフレー

4

リスト1　カメラ画像をアナログRGBディスプレイに動画表示するプログラム

```
#include<stdio.h>

#define        CPCTRL    (*((unsigned short *)0x00900000))
#define        SCTRL     (*((unsigned short *)0x00900002))
#define        SADDT     (*((unsigned short *)0x00900004))
#define        MISC      (*((unsigned short *)0x00900006))
#define        IOCTRL    (*((unsigned short *)0x00900008))
#define        LEDDLY    (*((unsigned short *)0x0090000A))

#define        MEM0      (*((unsigned short *)0x00A00000))
#define        MEM1      (*((unsigned short *)0x00A96000))
#define        MEM2      (*((unsigned short *)0x00B2C000))
#define        SIDOL     (*((unsigned short *)0xFFFF8622))
#define        SWAIT     (*((unsigned short *)0xFFFF8624))

#define        PACRL1    (*((unsigned short *)0xFFFF838C))
#define        PEDR      (*((unsigned short *)0xFFFF83B0))
#define        PEIOR     (*((unsigned short *)0xFFFF83B4))

#define        CMSTR     (*((unsigned short *)0xFFFF83D0))
#define        CMCSR0    (*((unsigned short *)0xFFFF83D2))
#define        CMCNT0    (*((unsigned short *)0xFFFF83D4))
#define        CMCOR0    (*((unsigned short *)0xFFFF83D6))

#define WCR1 (*((unsigned short *)0xFFFF8624))

int   i;
void wait();

void main(void)
{
  int i, k;
  int co3;
  unsigned short *SrcA;
  unsigned short *SrcB;
  unsigned short *DistA;
  unsigned short *DistB;
  unsigned short tmpus1, tmpus2;
  unsigned short rus1, rus2, rus3;
  short rshort1, rshort2, rshort3;
  unsigned short gus1, gus2, gus3;
  short gshort1, gshort2, gshort3;
  unsigned short bus1, bus2, bus3;
  short bshort1, bshort2, bshort3;
  unsigned short tmpshort;

  WCR1 = WCR1 & 0xF0FF | 0x0F00;  // WAIT = 15,  Memory cycle = 2+15

  PACRL1 = PACRL1 | 0x4000;       // MD15MD=1, CK=OUT

  PEIOR = PEIOR | 0xE1FF;         // default 0x0000, 1 is output, 0 is input, 1110 0001 1111 1111

      co3 = 0;

      SADDT=0x8012;               // カメラ・モジュールのリセット
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      SADDT=0x4011;               // 同期信号を設定；負論理
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      SADDT=0x1C12;               // RGB/Raw RGB出力モード，オート・ホワイト・バランス禁止
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      SADDT=0x111F;               // ＲＧＢ：５６５　出力
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      wait(75);                   // wait three frames

      if((PEDR&0x1200)==0x1200) { // PE12(SW2) is 1, PE9(SW1) is 1
               CPCTRL=0x0001;                      // capture to frame0, play frame0
               while((CPCTRL&0x0001)==1) {}
      } else if((PEDR&0x1200)==0x1000) { // PE12 is 1, PE9 is 0
               PEDR = 0x5555;     // LED ON
               CPCTRL=0x0009;                      // capture to frame0, play frame1
               while((CPCTRL&0x0001)==1) {}
               PEDR = 0xAAAA;     // LED OFF
               CPCTRL=0x010B;                      // subtract from frame0, capture to frame1, play frame1
```

リスト1　カメラ画像をアナログRGBディスプレイに動画表示するプログラム（つづき）

```
            while((CPCTRL&0x0001)==1) {}
        } else if((PEDR&0x1200)==0x0000) {    // PE12 is 0, PE9 is 0
        } else { // PE12 is 0, PE9 is 1
        }

        SADDT=0x0941;                         // flash off
        SCTRL=0x0001;
        while((SCTRL&0x0001)==1) {}

        if(co3 == 2) co3 = 0;
        else co3++;
   }
 }

 void wait(int w){
        CMCOR0 = 750-1;
        CMCNT0 = 0;
        CMCSR0 = 0x0001;                      // 24000000/32/750 = 1 mS
        CMSTR = 0x0001;                       // CMT0 start
        for(i=w; i>0; i--){
             while((CMCSR0 & 0x0080)==0)
                       ;
             CMCSR0 = 0x0001;
        }
        CMSTR = 0x0000;                       // CMT0 stop
 }
```

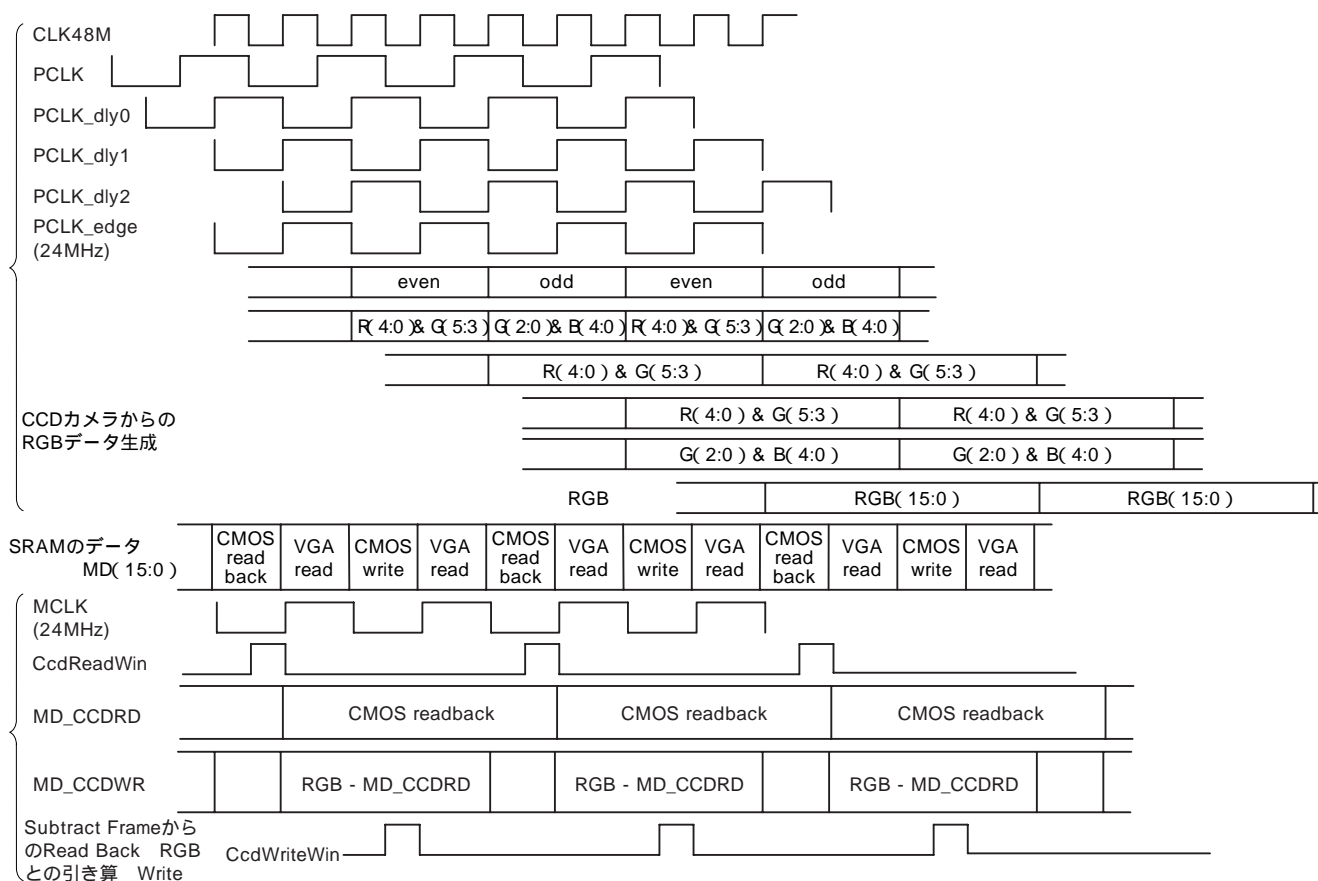



図7　フレーム・メモリ制御タイミング図

リスト2　カメラ入力画像の差分をアナログRGBディスプレイに表示するプログラム

```
#include<stdio.h>

#define        CPCTRL    (*((unsigned short
                                    *)0x00900000))
#define        SCTRL     (*((unsigned short
                                    *)0x00900002))
#define        SADDT     (*((unsigned short
                                    *)0x00900004))
#define        MISC      (*((unsigned short
                                    *)0x00900006))
#define        IOCTRL    (*((unsigned short
                                    *)0x00900008))
#define        LEDDLY    (*((unsigned short
                                    *)0x0090000A))

#define        MEM0      (*((unsigned short
                                    *)0x00A00000))
#define        MEM1      (*((unsigned short
                                    *)0x00A96000))
#define        MEM2      (*((unsigned short
                                    *)0x00B2C000))

#define        SIDOL     (*((unsigned short
                                    *)0xFFFF8622))
#define        SWAIT     (*((unsigned short
                                    *)0xFFFF8624))

#define        PACRL1    (*((unsigned short
                                    *)0xFFFF838C))
#define        PEDR      (*((unsigned short
                                    *)0xFFFF83B0))
#define        PEIOR     (*((unsigned short
                                    *)0xFFFF83B4))

#define        CMSTR     (*((unsigned short
                                    *)0xFFFF83D0))
#define        CMCSR0    (*((unsigned short
                                    *)0xFFFF83D2))
#define        CMCNT0    (*((unsigned short
                                    *)0xFFFF83D4))
#define        CMCOR0    (*((unsigned short
                                    *)0xFFFF83D6))

#define WCR1 (*((unsigned short *)0xFFFF8624))

int   i;
void wait();

void main(void)
{
  int i, k;
  int co3;
  unsigned short *SrcA;
  unsigned short *SrcB;
  unsigned short *DistA;
  unsigned short *DistB;
  unsigned short tmpus1, tmpus2;
  unsigned short rus1, rus2, rus3;
  short rshort1, rshort2, rshort3;
  unsigned short gus1, gus2, gus3;
  short gshort1, gshort2, gshort3;
  unsigned short bus1, bus2, bus3;
  short bshort1, bshort2, bshort3;
  unsigned short tmpshort;
  int CalEnd = 0;

  WCR1 = WCR1 & 0xF0FF | 0x0F00;
// WAIT = 15,  Memory cycle = 2+15

  PACRL1 = PACRL1 | 0x4000;
// MD15MD=1, CK=OUT
  PEIOR = PEIOR | 0xE1FF;                     // default
    0x0000, 1 is output, 0 is input, 1110 0001 1111 1111

      co3 = 0;

      SADDT=0x8012;
                              // カメラ・モジュールのリセット
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}
      SADDT=0x4011;
                                  // 同期信号を設定；負論理
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      SADDT=0x1C12;              // RGB/Raw RGB出力モード，
                                 オート・ホワイト・バランス禁止
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      SADDT=0x111F;
                                  // RGB：565.出力
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

      SADDT=0x2028;               // RGB
      SCTRL=0x0001;
      while((SCTRL&0x0001)==1) {}

  while(1){
      wait(75);                   // wait three frames

      if((PEDR&0x1200)==0x1200) { // PE12(SW2) is 1,
PE9(SW1) is 1
              if(CalEnd == 0) {
                      PEDR = 0x5555;     // LED ON
                      CPCTRL=0x0001;     // capture to
                                 frame0, play frame0
                      while((CPCTRL&0x0001)==1) {}
                      wait(1000);
                      PEDR = 0xAAAA;     // LED OFF
              } else {
                      CPCTRL=0x0000;
                                       // play frame0
              }
      } else if((PEDR&0x1200)==0x1000) {
                                  // PE12 is 1, PE9 is 0
              if(CalEnd == 0) {
                      CPCTRL=0x000B;
                     // capture to frame1, play frame1
                      while((CPCTRL&0x0001)==1) {}
                      wait(1000);
              } else {
                      CPCTRL=0x0008;
                                       // play frame1
              }
      } else if((PEDR&0x1200)==0x0200) {
                                  // PE12 is 0, PE9 is 1
      } else { // PE12 is 0, PE9 is 0
              if(CalEnd == 0) {
                      MISC=0x0020;
                                  // SRAM Access Enable

                      SrcA = &MEM0;
                      SrcB = &MEM1;
                      DistA = &MEM2;
                      for(i = 0; i < 480; i++) {
                      for(k = 0; k < 640; k++) {
                              tmpus1 =
                               *(SrcA + k);
                              tmpus2 =
                               *(SrcB + k);
                      // Red difference
                              rshort1 =
                       (short)(tmpus1 >> 11);
                              rshort2 =
                       (short)(tmpus2 >> 11);
                              rshort3 =
                           rshort1 - rshort2;
                              if(rshort3 <
                         0) rshort3 = 0;
                              rus3 =
(unsigned short)rshort3;
                              rus3 = rus3
<< 11;

                      // Green difference
                              gshort1 =
            (short)((tmpus1 >> 5) & 0x003F);
                              gshort2 =
            (short)((tmpus2 >> 5) & 0x003F);
```

リスト2　カメラ入力画像の差分をアナログRGBディスプレイに表示するプログラム（つづき）

```
                                gshort3 =
                          gshort1 - gshort2;
                                if(gshort3 <
                             0) gshort3 = 0;
                                gus3 =
                   (unsigned short)gshort3;
                                gus3 = gus3
                                        << 5;

                        // Blue difference
                                bshort1 =
                  (short)(tmpus1 & 0x001F);
                                bshort2 =
                  (short)(tmpus2 & 0x001F);
                                bshort3 =
                          bshort1 - bshort2;
                                if(bshort3 <
                             0) bshort3 = 0;
                                bus3 =
                   (unsigned short)bshort3;
                                bus3 = bus3;

                                *(DistA + k)
                     = rus3 + gus3 + bus3;
                        }
                        SrcA += 640;
                        SrcB += 640;
                        DistA += 640;
        }
```

```
                                CalEnd = 1;
                                MISC=0x0000;
// SRAM Access Disable
                        } else {
                                CPCTRL=0x0010;
// play frame2
                        }
                }

                SADDT=0x0941;           // flash off
                SCTRL=0x0001;
                while((SCTRL&0x0001)==1) {}

                if(co3 == 2) co3 = 0;
                else co3++;
        }
}

void wait(int w){
        CMCOR0 = 750-1;
        CMCNT0 = 0;
        CMCSR0 = 0x0001;        // 24000000/32/750 = 1 mS
        CMSTR = 0x0001;         // CMT0 start
        for(i=w; i>0; i--){
                while((CMCSR0 & 0x0080)==0)
                                ;
                CMCSR0 = 0x0001;
        }
        CMSTR = 0x0000;         // CMT0 stop
}
```

ムはビット10とビット9（Subtract Frame）で指定します．

CPUからピクセル・データをアクセスするときは，カメラと同じ，

- MCLK＝“L”のタイミング

で行います．

カメラ入力とCPUアクセスの切り替えは**図4（d）** Miscellaneousレジスタのビット5（SramAccessEnb）で行います．

実際の画像録画制御はFPGAが行うので，ユーザは**図4**の各レジスタに必要なデータを書き込むだけです．

8　画像処理実験のプログラム

リスト1はフレーム0にカメラ画像を取り込むと同時に，取り込んだ画像をVGA表示するためのプログラムです．開発にはイエローソフト社のコンパイラを使いました．CPUの仕事はカメラの初期設定とFPGAで構成する画像処理回路のレジスタ設定だけです．

30フレーム/sのカメラ・データを取りこぼすことなくキャプチャし，60フレーム/sでVGA表示します．表示レートが画像入力レートの2倍になるので，1コマごとに前後の画像がつなぎ合わさった画像が表示されます．

リスト2は複数のフレーム・メモリを活用した画像処理の実験プログラムです．**図8（a）**に示す「CPUによる差分演算処理」と**図8（b）**に示す「ハードウェアによる差分」を試してみました．

その結果，第1章で紹介したように，

- 差分データ＝動いている物体

が差分画像となって表示されます．厳密には動く物体の“抜けあと”が残るので，実用にはもう少し複雑なアルゴリズムを開発する必要があります．

ここで示したかったことは，

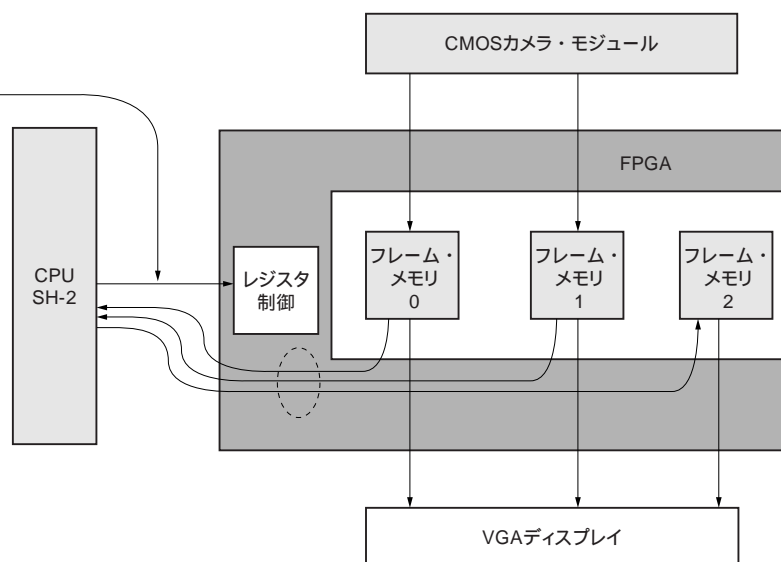
- CPUのピクセル処理で処理アルゴリズムを検証し
- 検証された処理アルゴリズムをハードウェアで高速化

するという手順です．画像処理アルゴリズム開発の効率化と工学的な実用化をスムーズに行えます．

9　画像ベースボード開発の狙い …画像アルゴリズムと画像処理IPのテスト・ベンチ

7月号付属基板の有効活用からスタートした画像ベースボードの開発ですが，今回のフレーム・メモリ搭載により多くの可能性が広がってきました．

画像処理回路は大量のデータを高速に処理することが求められます．CPLDやFPGAを使ったHDL回路設計のア

- PE12（SW2）が1でPE9（SW1）が1のとき
  ①を繰り返すようにレジスタを書く．
  このときは1秒ごとにフレーム0の静止画が更新される．
- PE12（SW2）が1でPE9（SW1）が0のとき
  ②を繰り返すようにレジスタを書く．
  このときは1秒ごとにフレーム1の静止画が更新される．
- PE12（SW2）が0でPE9（SW1）が0のとき
  キャプチャは停止され，CPUは③のようにフレーム0と1を読み，CPU内で差分をとり，結果をフレーム2に書き込み表示する（この間約2秒かかる）．その後フレーム2の静止画（差分）が表示される（更新なし）．
- （差分をとった後）PE12（SW2）が1でPE9（SW1）が1のときフレーム0の静止画が表示される（④，更新なし）．
- （差分をとった後）PE12（SW2）が1でPE9（SW1）が0のときフレーム1の静止画が表示される（⑤，更新なし）．
- （差分をとった後）PE12（SW2）が0でPE9（SW1）が0のときフレーム2の静止画が表示される（⑥，更新なし）．
- 再度差分をとるにはCPUをリセットする．

（**a**）still_hello.c（静止画モード，CPUで静的にフレーム差分をとる）

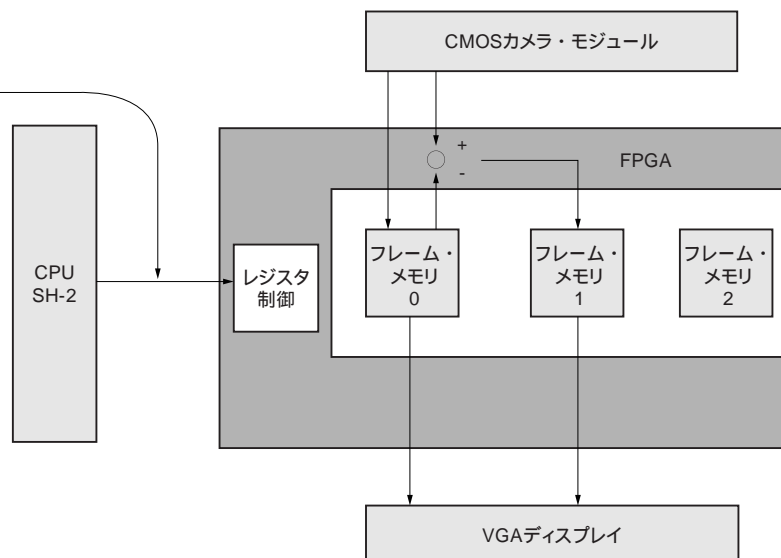
- PE12（SW2）が1でPE9（SW1）が1のとき
  ①を繰り返すようにレジスタを書く．
  このときは普通の動画が見える．
- PE12（SW2）が1でPE9（SW1）が0のとき
  ②，③，②，③，…と繰り返すようにレジスタを書く．
  FPGA内で動的に差分がとられ表示される．
- レジスタを書く間隔は2フレーム分（66.7ms）なので
  この間隔で差分が更新・表示される．

（**b**）hello.c（動画モード，FPGAで動的にフレーム差分をとる）

図8　CPUとFPGによる複数のフレーム・メモリを活用した画像処理の実験プログラム

プリケーションとして真価を発揮できる分野です．

設計結果をすぐに目で確認できるので論理設計の教育ツールとしても有効です．最新のデジタルCMOSカメラやTFT液晶表示モジュールなどの入手ルートも開拓しました．教育，開発，研究用テスト・ベンチとして活用されることを期待しています．

えさき・まさやす
（株）イーエスピー企画
代表取締役